RESEARCH PAPER

Scaling Notebooks as Re-configurable Cloud Workflows

Yuandou Wang[1†], Spiros Koulouzis[1,2], Riccardo Bianchi[1,2], Na Li[1], Yifang Shi[2,3], Joris Timmermans[2,3], W. Daniel Kissling[2,3] & Zhiming Zhao[1,2†]

[1]Multiscale Networked Systems, Informatics Institute, University of Amsterdam, 1098XH Amsterdam, The Netherlands

[2]LifeWatch ERIC, Virtual Lab & Innovation Center (VLIC), Science Park 904, 1098XH Amsterdam, The Netherlands

[3]Institute for Biodiversity and Ecosystem Dynamics (IBED), 1098XH Amsterdam, The Netherlands

Keywords: Scientific experiments; Jupyter Notebooks; Workflow management; Ecosystem structure data products; Cloud; Scalability

Citation: Wang, Y.D., et al.: Scaling notebooks as re-configurable cloud workflows. Data Intelligence 4(2), 409-425 (2022). doi: 10.1162/dint_a_00140

Received: September 19, 2021; Revised: January 1, 2022; Accepted: March 1, 2022

ABSTRACT

Literate computing environments, such as the Jupyter (i.e., Jupyter Notebooks, JupyterLab, and JupyterHub), have been widely used in scientific studies; they allow users to interactively develop scientific code, test algorithms, and describe the scientific narratives of the experiments in an integrated document. To scale up scientific analyses, many implemented Jupyter environment architectures encapsulate the whole Jupyter notebooks as reproducible units and autoscale them on dedicated remote infrastructures (e.g., high-performance computing and cloud computing environments). The existing solutions are still limited in many ways, e.g., 1) the workflow (or pipeline) is implicit in a notebook, and some steps can be generically used by different code and executed in parallel, but because of the tight cell structure, all steps in the Jupyter notebook have to be executed sequentially and lack of the flexibility of reusing the core code fragments, and 2) there are performance bottlenecks that need to improve the parallelism and scalability when handling extensive input data and complex computation.

In this work, we focus on how to manage the workflow in a notebook seamlessly. We 1) encapsulate the reusable cells as RESTful services and containerize them as portal components, 2) provide a composition tool for describing workflow logic of those reusable components, and 3) automate the execution on remote cloud infrastructure. Empirically, we validate the solution's usability via a use case from the Ecology and

† Corresponding authors: Yuandou Wang (Email: y.wang8@uva.nl; ORCID: 0000-0003-4694-9572) and Zhiming Zhao (Email: z.zhao@uva.nl; ORCID: 0000-0002-6717-9418).

Earth Science domain, illustrating the processing of massive Light Detection and Ranging (LiDAR) data. The demonstration and analysis show that our method is feasible, but that it needs further improvement, especially on integrating distributed workflow scheduling, automatic deployment, and execution to develop as a mature approach.

1. INTRODUCTION

The study of many scientific problems, e.g., significant environmental challenges or cancer diagnosis, requires large data volumes, advanced modeling techniques, and distributed computing facilitates [1, 2]. The literate computing environments such as Jupyter notebook, JupyterLab, and JupyterHub have exploded in popularity and emerged as a de-facto standard across different engineering and science domains [4, 5, 6], e.g., ecology [10, 11], biology [12], and medical research [13, 14]. They provide an excellent combination of explanatory text, software code, computational output, and multimedia resources in an executable interactive document, in which users input programming code or text in rectangular cells in the Jupyter notebook.

The narratives of the scientific experiment in a notebook can be described as scientific pipeline steps or workflow contained with executable code fragments. However, the workflow usually is implicit in a notebook without an explicit, structured workflow-oriented description. For instance, some steps can be represented as atomic tasks in conjunction with the dependencies regarding inputs/outputs. Still, because of the tight cell structure of the notebook, the workflow patterns cannot be well extracted, which further influences the reusability of some generic code fragments [24]. Besides, from the perspective of input/output dependencies, there are still some performance bottlenecks in the workflow execution, e.g., some steps need to be parallelized or scaled out, especially when the inputs are a large volume of data.

Current approaches usually enable the computational notebook as the whole job to be scaled out on a dedicated remote infrastructure, e.g., high-performance computing (HPC) settings and cloud environments [7, 15, 16, 17, 18, 19, 20, 21]. Generally, to bridge the gap between exploratory scientific analysis and computing environments, many implemented Jupyter environment architectures are mainly coupled with a pre-configurable infrastructure (e.g., via IaaS Cloud) to dynamically deploy and manage notebook-based application instances. However, widely-used solutions ignored the workflow-oriented representation and management in the single notebook (i.e., workflow structure regarding internal dependencies and efficient parallelism for task's input sizes). As a result, managing such a workflow in the notebook becomes a major bottleneck to meet the demands of scaling scientific experiments from domain researchers.

In light of this, we are motivated by the research question: how to seamlessly manage workflow in a notebook? In this paper, we give the first answer to this question and propose the Notebook-as-a-Workflow (NaaW) method. Our contributions mainly include:

1. we propose and prototype the component containerizer, which can encapsulate the reusable code fragments (cells) as RESTful services and containerize them as science portal components.

2. we provide a workflow composition tool for presenting the workflow logic of those reusable components with visualization.
3. we integrate the infrastructure automation tool to automate the workflow execution on remote cloud infrastructure.

The rest of the paper is organized as follows. Section 2 states the problem with a use case from the ecology and earth science domain, including the challenges and requirements, a review of the existing work regarding the topic of scaling scientific applications on distributed computing infrastructures, and a summary of the limitations of existing solutions. Section 3 provides the prototype design of our method and implementation details of our prototypes; in Section 4, we demonstrate the prototype from an example of processing massive Light Detection and Ranging (LiDAR) data and discuss the limitations of current work, and, finally, Section 5 concludes this paper and points out future research directions.

2. PROBLEM STATEMENT AND RELATED WORK

2.1 Problem Statement

Consider the example from the ecology and earth science domain, where researchers aim to monitor the changes of ecosystem structure over time or derive metrics of vegetation structure to model animal distributions and habitat suitability. For this purpose, they process raw data (3D point clouds) from country-wide, airborne Light Detection and Ranging (LiDAR) dataset (~16TB), using the Laserfarm workflow [11] to generate LiDAR-derived metrics related to ecosystem height, ecosystem cover, and ecosystem structural complexity. Two examples of statistical metrics that can be derived from LiDAR point clouds are shown in Figure 1, i.e., the 95th percentile of normalized height (as a measure of ecosystem height) and pulse penetration ratio (as a measure of ecosystem cover). The 95th percentile of normalized height quantifies the vegetation height of tallest plants in a given grid cell (e.g., trees in a forest patch) whereas the pulse penetration ratio describes vegetation openness as the ratio between the number of ground points relative to the total number of points within a given grid cell. By measuring them in two time periods from two country-wide LiDAR datasets, researchers can derive measures of ecosystem structural change. It is achieved by extracting statistical properties of LiDAR point clouds that are provided by Airborne Laser Scanning (ALS) surveys, usually in LAS/LAZ format.

Challenges. Researchers design algorithms and prototypes using a Jupyter environment such as a Jupyter notebook and conduct the scientific experiment on the local experimental platform or a small cluster. Those studies rely on appropriate algorithms, but even more on large-scale data, e.g., handling multi-terabyte datasets. However, the local compute and storage capacities might be inefficient in conducting large-scale scientific analysis, e.g., it is burdened by large-scale data inputs and the need for efficient and scalable computing. Besides, the notebook's pipeline steps or component modules can be presented as a workflow, as shown in Figure 2. Some modules as generic algorithms can be reused by different code and parallelized and scaled out on dedicated remote infrastructures, e.g., Cloud environments. Nevertheless, managing such a workflow in the Jupyter notebook is still lacking. Issues of the (un)expected overheads or performance

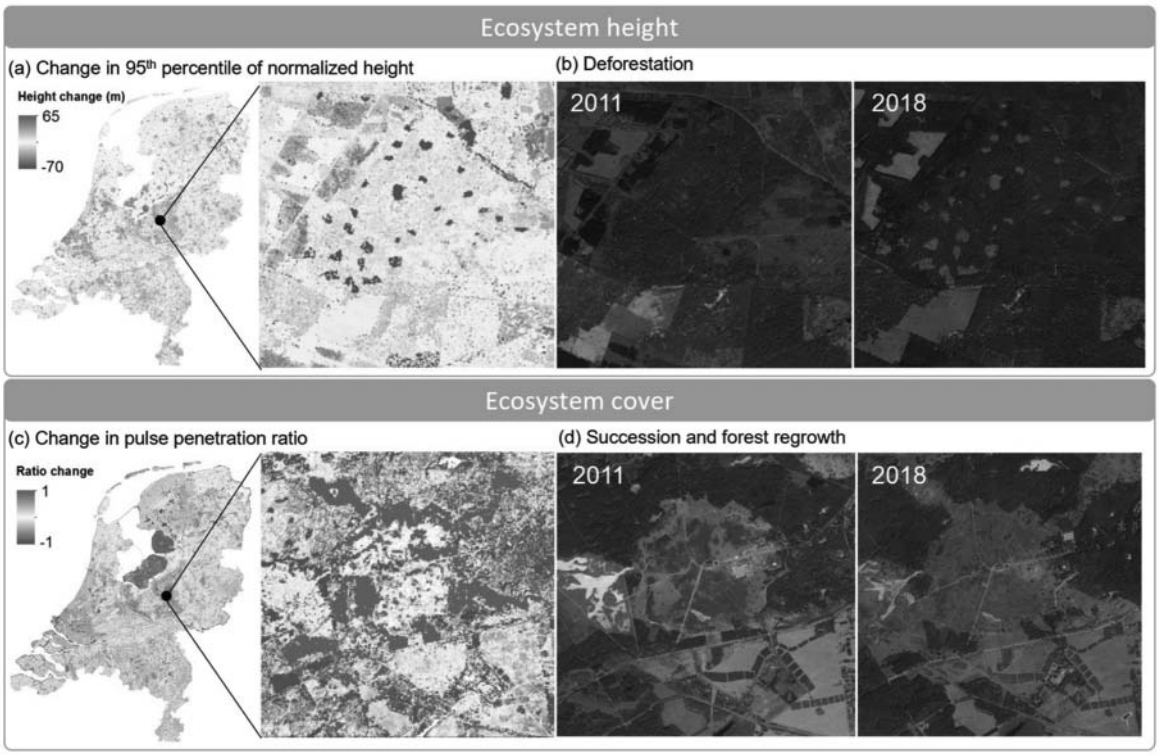

Figure 1. Two examples of statistical metrics derived from airborne LiDAR point clouds across the Netherlands to measure changes in ecosystem structure, including a visual comparison with images from Google Maps. (a, b) Changes in ecosystem height (measured from LiDAR point clouds with a metric called 95th percentile of normalized height) to detect deforestation. The blue color indicates the decrease of vegetation height (i.e., 95th percentile of normalized height) from 2011 to 2018, while the red color indicates the opposite scenario. (c,d) Changes in ecosystem cover (measured from LiDAR point clouds with a metric called pulse penetration ratio) to map the succession and re-growth of forests). The red color indicates the increase of pulse penetration ratio from 2011 to 2018, describing the vegetation cover has decreased during this time period, while the blue color indicates the opposite scenario.

bottlenecks caused by data transfer between containerized components in the NaaW are outside the scope of this paper; however, we indeed have considered this problem. When processing the large volume data with approximately 16TB, we are using a splitter module and a merger module to cope with this issue at present. The splitter module can partition the large data volume into multiple more minor data predefined by the user to remove the large data volume transfer between containerized components. The merger module is responsible for merging these distributed processed results. As a result, it does not involve very frequent transmissions of enormous data volume between containerized components; on the contrary, it improves the parallelism of data processing.

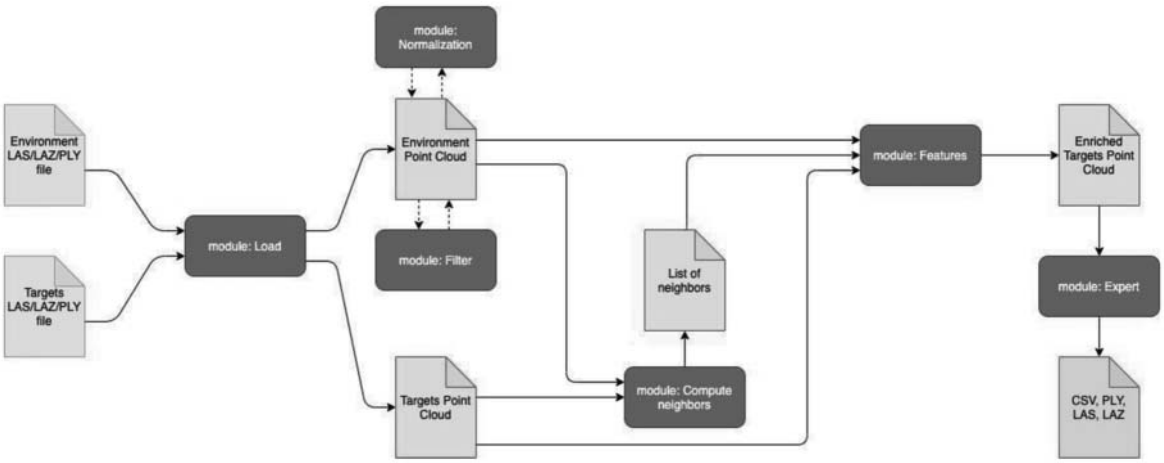

Figure 2. The default workflow in the Jupyter notebook using the example of the Laserchicken software [25]. LiDAR datasets (e.g., in LAS/LAZ/PLY format) are loaded and a set of target points define the environment point cloud (EPC) and the target point cloud (TPC), respectively. The LiDAR point clouds are normalized, neighbors are computed and features (i.e., statistical properties of the point cloud) are subsequently extracted over neighbors and appended to the associated target points. This forms the enriched target point cloud (eTPC), which can be exported in several formats. Optionally, the EPC can be filtered before further processing [10].

2.2 Related Work

Many implemented Jupyter environment architectures are mainly based on HPC settings and cloud environments [15, 16, 18, 19, 20, 21]. For instance, the work of Milligan *et al.* [15, 16] implemented the classic architectures on which researchers integrated the Jupyter platform with supercomputing resources to bridge the gap between exploratory data analysis and HPC, especially leverage Jupyter for interactive data-intensive supercomputing services. Their solutions offer HPC notebook service, a science portal for the GEMS platform for argo-informatics data sharing and analysis, and BinderHub services. The HPC notebooks service implemented at MSI (Minnesota Supercomputing Institute) permits the user to seamlessly run the interactive Jupyter notebook web application using normal batch-scheduled clustered computing resources. The BinderHub is a component for automatically launching Jupyter-ready Docker containers built by JupyterHub with repo2docker within a Kubernetes cluster using public cloud resources. Likely, Zonca et al. [19] proposed three deployment strategies for scaling up scientific applications on XSEDE resources at different levels of scalability, i.e., JupyterHub on HPC via the batch scheduler, JupyterHub on XSEDE Jetstream with Docker Swarm mode and with Kubernetes. The differences between deploying Docker in Swarm mode and Kubernetes mainly reflect the former providing notebooks with persistent storage and quota while the latter provides a fault-tolerant JupyterHub deployment with elasticity.

Henderson et al. [18] proposed their NERSC (National Energy Research Scientific Computing Center) Jupyter infrastructure and JupyterHub deployment model. They use notebooks as reusable curated recipes or applications (i.e., having the ability to run the notebook on different data or with varying inputs without

copying or editing the notebook each time) to simplify the execution processes. Although the work of Henderson *et al.* [18] is close to NaaW since it also has the potential for scaling up different data inputs in the notebook and running these steps in parallel across multiple HPC nodes, it differs heavily from NaaW. Because it is based on NERSC HPC resources instead of a cloud environment with on-demand resource provisioning, it also performs limitations of the flexibility to reuse critical components in a notebook as part of a distributed workflow.

Yin et al. [17] proposed the CyberGIS-Jupyter framework for scalable geospatial analytics. The alternative solution adopts Jupyter notebooks instead of web GIS as the front-end interface to provide a consistent and agile playground for both developers and users. It uses JupyterHub and Docker Swarm as the CyberGIS-Jupyter server. Encapsulating CyberGIS capabilities within a pre-configured and containerized environment achieves on-demand resource provisioning through Openstack to elastically deploy and manage multiple virtual machine (VM) instances of the applications. The hybrid computing environment called ROGER integrated HPC and cloud resources offers the underlying infrastructure support for the reproducible deployment. Nevertheless, their approach still has several limitations related to the scientific workflow representation. For instance, their work did not emphasize pipeline or workflow. Also, the notebook performed like the whole job instead of a workflow-oriented application to be packaged out and submitted onto the CyberGIS-Jupyter cloud environment.

Most of the existing solutions focus on the execution of the notebook on a pre-configured infrastructure, e.g., HPC cluster or virtual machines provisioned in the cloud. In those solutions, there is no explicit workflow management in the notebook; users control the execution of the experiment steps via the cells in the notebook. The workflow logic and data flows are implicitly described in the order of the cells in the notebook, which may hamper the reconfiguration of the notebook at the cell level for different purposes. We aim to overcome those limits by providing extra Jupyter extensions to extract the notebook cells as reusable components, to compose new logic, and to automate the execution on remote infrastructure. Those extensions cover the key steps of workflow management.

3. MANAGING NOTEBOOK AS A WORKFLOW

3.1 Requirements

To tackle the above challenges and gaps, we identify the following requirements for managing the workflow in notebooks and provide basic design ideas for our solution named NaaW.

- The ***workflow management process***, e.g., the decomposition of a single notebook containing code fragments and workflow composition, must be flexible alongside a native Jupyter environment for scientific research. Users should not be restricted in their code fragments' selection and workflow design choices for their experiments.

- The workflow building blocks, i.e., the reusable components from the notebooks, must be interoperable with self-defined workflow logic. The tool should enable users to select those reusable components to construct scientific pipelines and guarantee the dependency constraints between components.
- The approach must provide scalable solutions for large-scale scientific experimental analysis, especially for large datasets or complicated computation demands, and automatically execute the workflow on remote infrastructures such as the cloud. The scientific experimental process must be scalable and parallel to solve performance bottlenecks.
- The solutions should be embedded as part of the current Notebook ecosystem so that the scientists do not have to drop the advantages of the native Jupyter.

3.2 Proposed Jupyter Extensions

Based on the requirements, we proposed four Jupyter extensions.

Component Containerizer creates reusable workflow building blocks from the notebook cells. Technical details of this extension have been discussed in our earlier paper [22]. This Jupyter extension allows users to select code cell(s) from Jupyter Notebooks effectively and generate reusable workflow building blocks (REST services) in a WYSIWYG manner. The extension can also containerize the services as self-contain deployable containers (i.e., Docker) and store them in a local catalog or remote Docker Hub.

Experiment Manager allows users to compose workflows using containerized notebook code (using the component containerizer). Users can compose dependencies among different blocks and construct distributed workflows by visually connecting the input and output parameters within the metadata description. This extension can save the generated workflow as workflow specification documents such as YAML or CWL syntax. For example, Figure 3 presents a straightforward vision of the experiment manager usage.

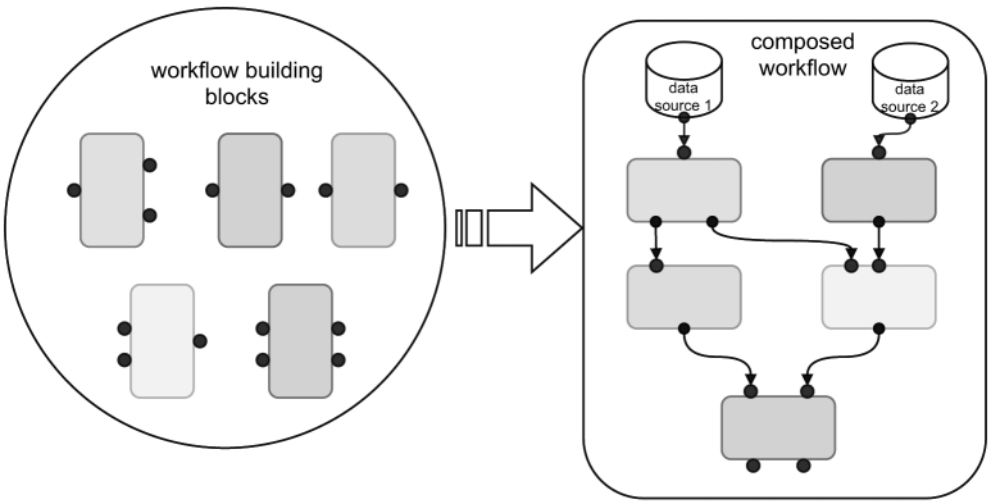

Figure 3. A conceptual diagram of the experiment manager extension.

Distributed Workflow Bus plans and schedules the workflow deployment and execution on remote infrastructures. The automation of the infrastructure services provisioning and deployment will be provided by the extension called remote infrastructure automator, to be discussed next. This extension enables users to submit the workflow specifications (created by the experiment manager), together with input data sources, resource budget (for using cloud services). The workflow bus first invokes the remote infrastructure automator to initialize the run-time infrastructure and then uses its built-in scheduling algorithms to schedule the workflow execution on the infrastructure, as shown in Figure 4. Currently, this function is built atop the Argo workflow engine, enabling developers to select built-in scheduler policies or even add their self-defined scheduling strategies.

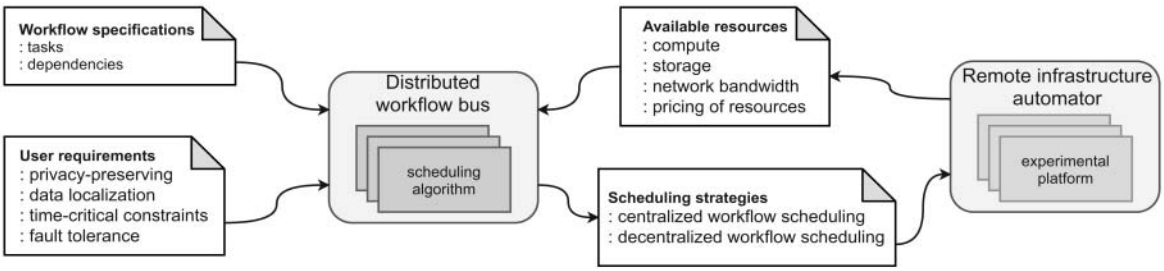

Figure 4. The data flow of the distributed workflow bus and remote infrastructure automator.

Remote Infrastructure Automator plans the cloud infrastructure capacity and automates the resource provisioning and service deployment. This extension is extended from earlier work of Dynamic real-time infrastructure planner [24].

3.3 How do They Work Together

The four key components will be installed as Jupyter extension, as shown in Figure 5. The grey boxes present the four key components, i.e., the component containerizer, experiment manager, distributed workflow bus, and remote infrastructure automator. We also highlight the catalog that contains workflow building blocks and the dedicated remote infrastructure for scalable experiments.

Users prototype scientific pipeline steps using Jupyter environments (e.g., Notebooks) to conduct their experiments at a small scale, e.g., on a local computer or a small cluster. Based on the native Jupyter notebook front-end, users can use the component containerizer to encapsulate a cell as a REST service and containerize it (step 1), store the containerized components in a local catalog (step 2), and make further changes to the components (step 3). Using the experiment manager extension, the user can design a new workflow experiment (step 4) by selecting containerized components from the local catalog (step 5) and create a representation of the workflow (step 6). The workflow description will be executed by the distributed workflow bus (step 7) by first initializing virtual infrastructures (e.g., VMs or Kubernetes cluster) from providers (step 9) via the remote infrastructure automator extension. The runtime status and the results can be monitored by the workflow bus via a dashboard (step 10).

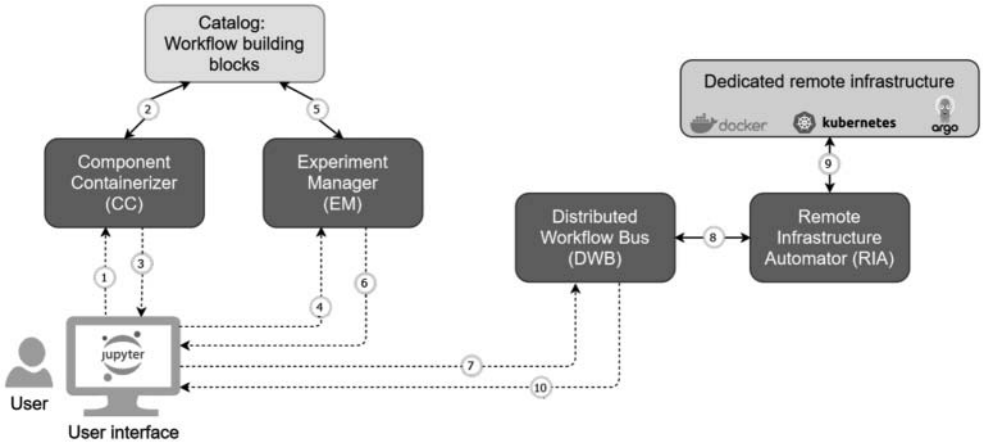

Figure 5. The conceptual model of our proposed solution.

4. SYSTEM PROTOTYPE AND DEMONSTRATION

As discussed before, Figure 2 presents a sample workflow of the Laserchicken software from the ecology domain, which mainly consists of six functional modules, including the load, normalize, filter, compute neighbors, features, and export, each of them has different code fragments with additional input and output parameters. Practically, the whole process is placed and has to be run sequentially with a tight cell structure in the notebook, which strongly influences the performance and scalability of the experiment, e.g., reducing the parallelism and reusability of the code fragments. To validate our method, we demonstrate the critical components of workflow-oriented management in a Jupyter notebook environment using the above use case.

Figure 6 presents the interface of the component containerizer alongside a Jupyter notebook. The single notebook contains scientific pipeline steps with different component functions. Users can select the code fragments to encapsulate them as a component. And as shown in the left bar, it also automatically inspects the corresponding metadata, e.g., inputs, outputs, parameters, and package dependencies for each component encapsulation. And as workflow building blocks, they can be added to the catalog for further usage.

Figure 7 shows the function of experiment manager. The local catalog contains the metadata of encapsulated components. Users can select different components as reusable workflow building blocks to compose self-defined workflow logic, e.g., tasks and dependencies, and export the workflow. The desirable output of the experiment manager is the generated workflow specification document, which is one of the inputs of the distributed workflow bus module.

As shown in Figure 8, currently, the distributed workflow bus is implemented atop the Argo workflow engine for Kubernetes. It uses the default scheduler for the workflow planning and scheduling; the underlying infrastructure, such as virtual machines, still manually allocate virtual machines for launching the Kubernetes platform.

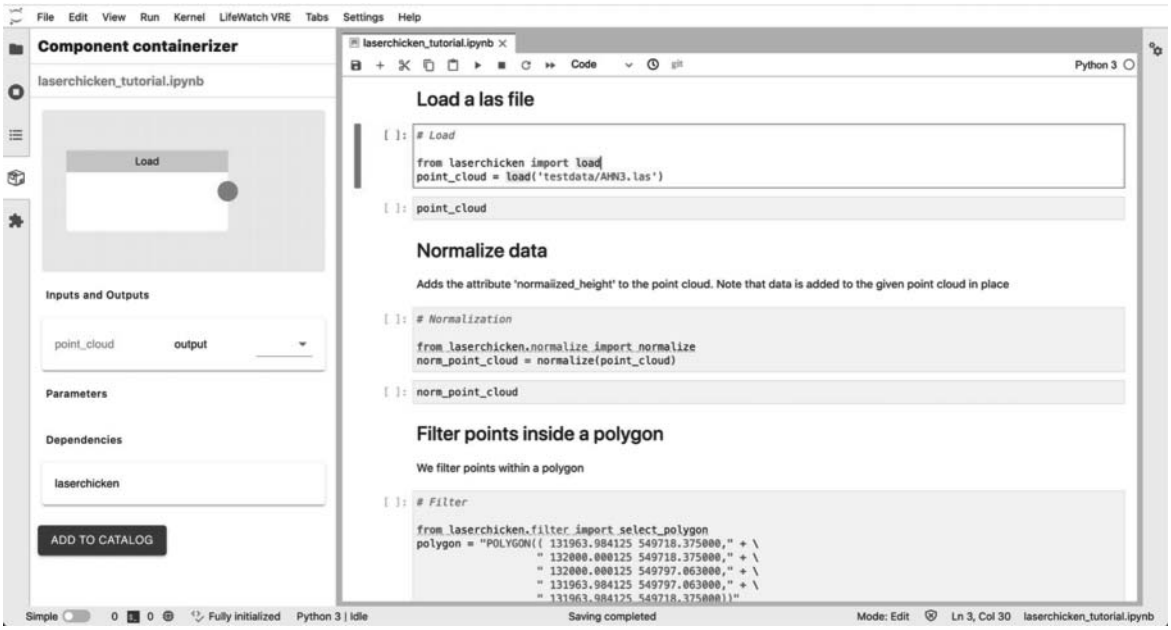

Figure 6. The interface of the component containerizer modular.

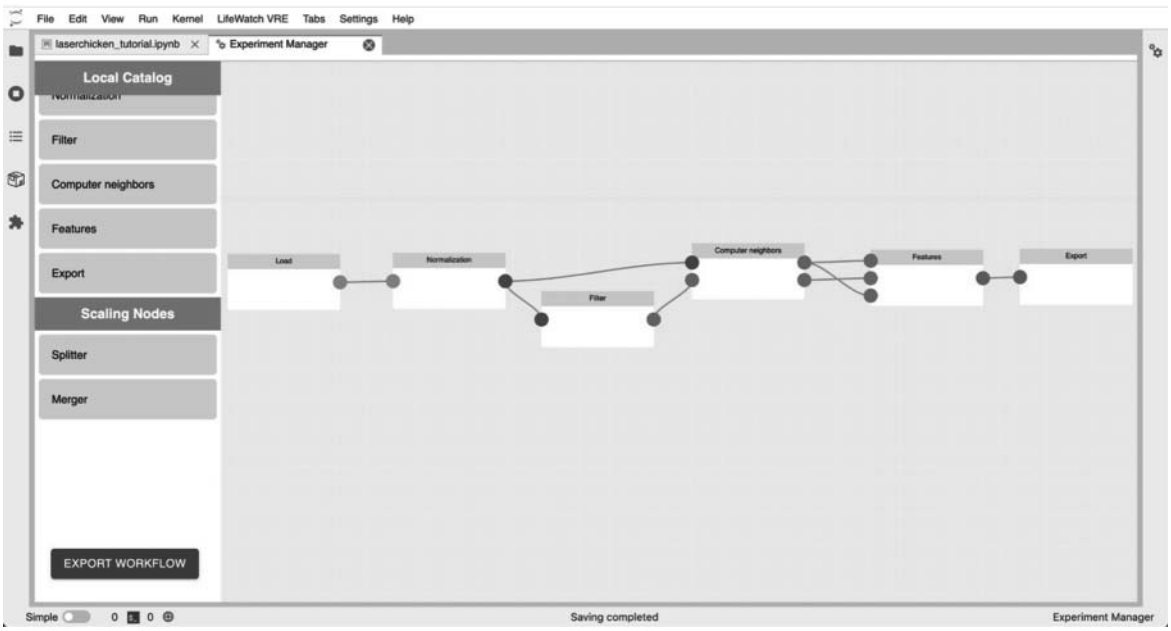

Figure 7. The interface of the experiment manager modular.

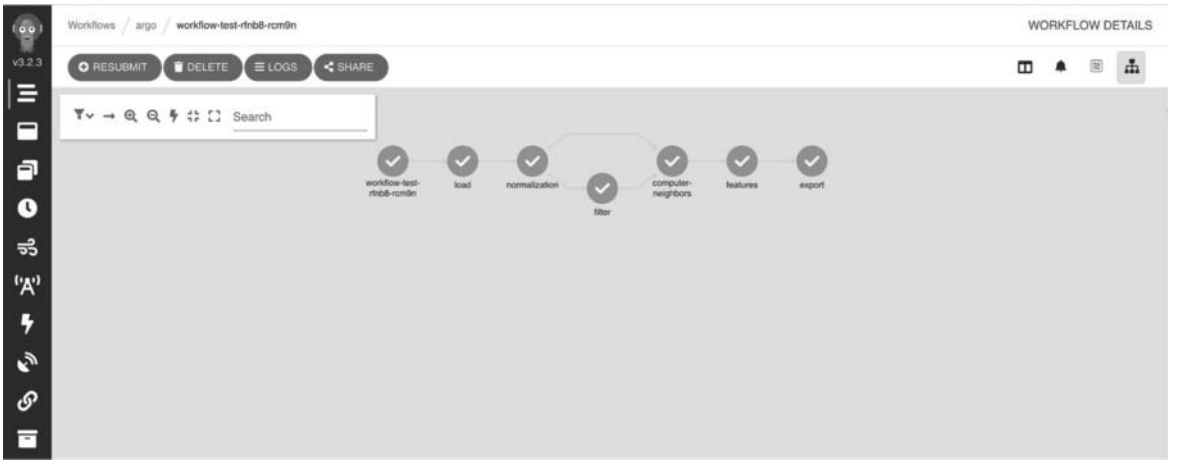

Figure 8. The current distributed workflow bus solution with Argo workflow engine.

Figure 9 shows the interface of the remote infrastructure automator (also called Cloud-cells). This Jupyter notebook extension allows the user to deploy dockers generated by Cloud-Cells to the cloud. But it has not yet been completely integrated into the NaaW method.

4.1 Discussion

In this paper, we discussed the key components in Notebook-as-a-Workflow (NaaW) solution. The four components are prototyped as extensions embedded in the Jupyter working environment of the scientists. This solution extends some of our previous work [22, 24] and further extends to a workflow-oriented management approach to bridge the gap between the Jupyter environment and workflow management at scale. By demonstrating via a LiDAR processing example from the ecology and earth science domain, the proposed solution can achieve the workflow management process mentioned in the requirements. Using those Jupyter extensions, a scientist can 1) interactively encapsulate code fragments (one or more notebook cells) in the Jupyter notebook as reusable workflow components, 2) compose a workflow using those workflow components and customize the execution logic based on data volume and locations, 3) automate the cloud infrastructure based on the workflow requirements (data volume, and resource budgets), and 4) interactively execute the composed workflow on the cloud infrastructure to achieve the required scale. In the paper, we only focus on the key steps in workflow management, e.g., workflow component containerization from the notebook, workflow composition, infrastructure automation, and workflow execution. There are still several features that have not yet been implemented or can be improved. For instance, the experiment manager tool could not inspect the correctness of dependencies between two workflow building blocks. It needs an inspection before users save their generated workflow specifications and submit them to the distributed workflow bus module. In addition, the design and implementation of distributed workflow bus are not developed maturely now since it is still based on a third party, e.g., the Argo workflow engine. Many workflow scheduling algorithms are not well developed as well, e.g., we still use the default scheduler of the Argo workflow engine to deploy and execute the submitted workflows. The

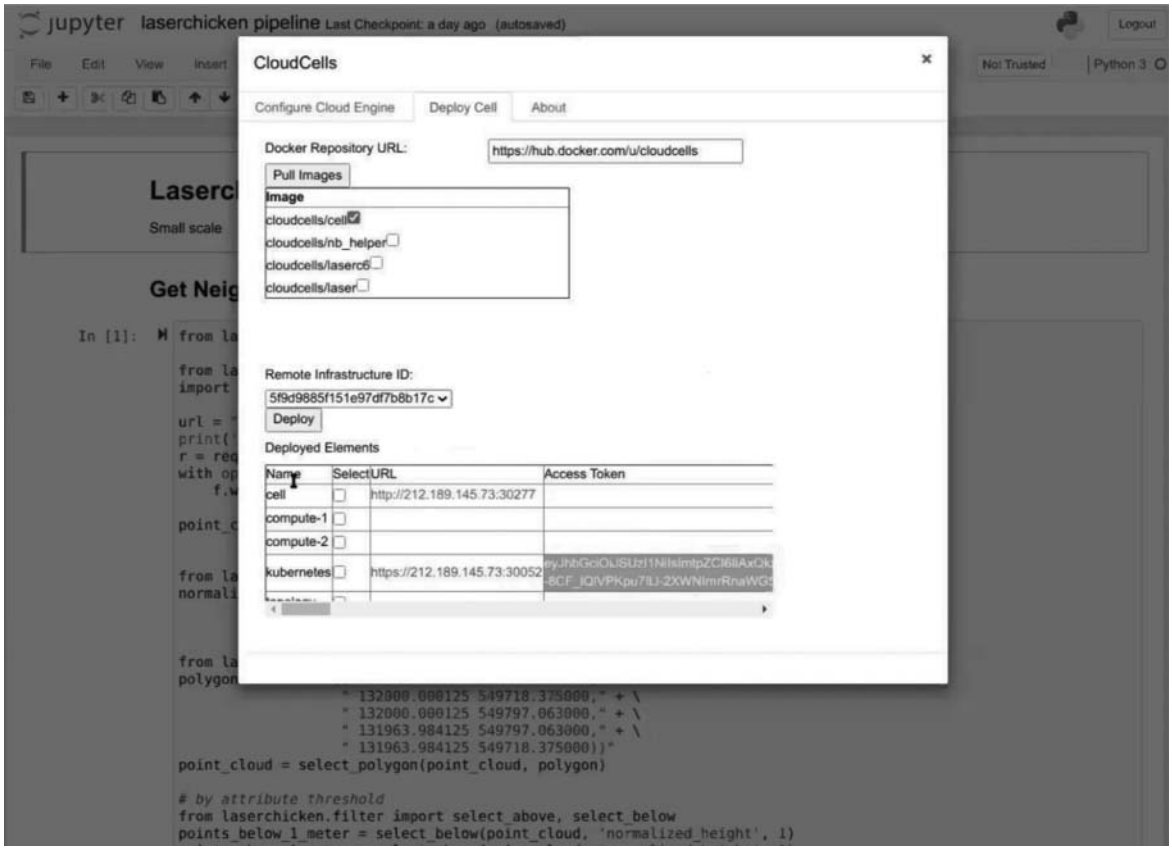

Figure 9. The remote infrastructure automator.

integration of distributed workflow bus and remote infrastructure automator does not work seamlessly. These two modules could not cooperate smoothly though both of them as Jupyter extensions can be built-in Jupyter environment. Some steps have to be completed manually.

5. CONCLUSION AND FUTURE WORK

In this work, we focus on managing the workflow in a Jupyter notebook architecture. We propose four core components to achieve our goal, i.e., the component containerizer, experiment manager, distributed workflow bus, and remote infrastructure automator, respectively. We 1) encapsulate the reusable cells as RESTful services and containerize them as portal components in the component containerizer module, 2) provide a composition tool for describing workflow logic of those reusable components, and 3) automate the execution on remote cloud infrastructure according to the distributed workflow bus and remote infrastructure automator. We validate the usability of the solution via a LiDAR use case from the ecology and earth science domain. This work is still developing and in continuous improvement. The missing features mentioned in the discussion section will be on our agenda for the next steps.

ACKNOWLEDGEMENTS

This work has been partially funded by the European Union's Horizon 2020 research and innovation programme by the project CLARIFY under the Marie Sklodowska-Curie grant agreement No 860627, by the ARTICONF project grant agreement No 825134, by the ENVRI-FAIR project grant agreement No 824068, by the BLUECLOUD project grant agreement No 862409, by the LifeWatch ERIC.

AUTHOR CONTRIBUTIONS

Y.D. Wang (y.wang8@uva.nl): Conceptualization, Investigation, Writing—original draft, Writing—reviewing & editing.

S. Koulouzis (S.Koulouzis@uva.nl): Conceptualization, Software development and maintenance.

Riccardo Bianchi (r.bianchi@uva.nl): Conceptualization, Software development and maintenance.

N. Li (n.li@uva.nl): Discussion.

Y.F. Shi (y.shi@uva.nl): Writing—review & editing, Visualization.

J. Timmermans (j.timmermans@uva.nl): Conceptualization, Writing—original draft, Writing—review & editing.

W.D. Kissling (wdkissling@gmail.com): Conceptualization, Writing—original draft, Writing—review & editing, Visualization, Funding acquisition.

Z.M. Zhao (Z.Zhao@uva.nl): Conceptualization, Writing—original draft, Writing—review & editing, Supervision, Project administration, Funding acquisition, Coordinated the technical development of the system.

REFERENCES

[1] Vermeulen, A., et al.: Supporting cross-domain system-level environmental and earth science. In: Zhao, Z., Hellström, M. (eds.) Towards Interoperable Research Infrastructures for Environmental and Earth Sciences, pp. 3–16. Springer, Cham (2020)
[2] Tuli, S.: Next generation technologies for smart healthcare: Challenges, vision, model, trends and future directions. Internet Technololy Letters 3(2), e145 (2020)
[3] Tudoran, R., et al.: Adaptive file management for scientific workflows on the Azure cloud. In: 2013 IEEE International Conference on Big Data, pp. 273–281 (2013)
[4] Jupyter. Available at: https://jupyter.org/. Accessed 11 March 2020
[5] Jupyter hub. Available at: https://jupyter.org/hub. Accessed 11 March 2020
[6] Perkel, J.M.: Why Jupyter is data scientists' computational notebook of choice. Nature 563(7729), 145–146 (2018)

[7] Henderson, M. L., et al.: Accelerating experimental science using Jupyter and NERSC HPC. In: Juckeland, G., Chandrasekaran, S. (eds.) Tools and Techniques for High Performance Computing, pp. 145–163. Springer International Publishing, Cham (2020)
[8] LifeWatch virtual lab and innovation center. Available at: https://www.lifewatch.eu/vlabs-innovations-centre. Accessed 11 March 2020
[9] The EU ENVRI-FAIR project. Available at: http://www.envriplus.eu/. Accessed 11 March 2020
[10] Meijer, C., et al.: Laserchicken—A tool for distributed feature calculation from massive LiDAR point cloud datasets. SoftwareX 12, 100626 (2020)
[11] Laserfarm. Available at: https://github.com/eEcoLiDAR/Laserfarm/. Accessed 11 March 2020
[12] BioExcel. Available at: https://mmb.irbbarcelona.org/biobb/. Accessed 13 September 2021
[13] Deep histopath. Available at: https://github.com/CODAIT/deep-histopath. Accessed 13 September 2021
[14] Building medical 3D image segmentation using Jupyter notebooks from the NGC catalog. Available at: https://developer.nvidia.com/blog/building-medical-3d-image-segmentation-using-jupyter-notebooks-from-the-ngc-catalog/. Accessed 13 September 2021
[15] Milligan, M.: Interactive HPC gateways with Jupyter and Jupyterhub. In: PEARC17: Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact, Article No. 63 (2017)
[16] Milligan, M.B.: Jupyter as common technology platform for interactive HPC services. In: PEARC '18: Proceedings of the Practice and Experience on Advanced Research Computing, Article No. 17 (2018)
[17] Yin, D., et al.: CyberGIS-Jupyter for reproducible and scalable geospatial analytics. Concurrency Computation 31(11), e5040 (2019)
[18] Henderson, M.L., et al.: Accelerating experimental science using Jupyter and NERSC HPC. In: HUST 2019, SE-HER 2019, WIHPC 2019: Tools and Techniques for High Performance Computing, pp. 145–163 (2019)
[19] Zonca, A., Sinkovits, R.S.: Deploying Jupyter notebooks at scale onXSEDE resources for science gateways and workshops. arXiv preprint arXiv:1805.04781 (2018)
[20] Stubbs, J., et al.: Integrating Jupyter into research computing ecosystems. In: PEARC '20: Practice and Experience in Advanced Research Computing, pp. 91–98 (2020)
[21] Zhou, S., Liu, X., Lin, L.: Design and implementation of Python teaching platform based on container and Jupyter. In: ACM International Conference Proceeding Series, pp. 446–450 (2020)
[22] Kruijer, W., et al.: FAIR-Cells: An interactive tool for enabling the FAIRness of code fragments in Jupyter notebooks. In: The proceedings of International Conference of High Performance Computing and Simulation (HPCS) (2020). (To appear)
[23] Cloud-Cells. Available at: https://github.com/QCDIS/Cloud-Cells. Accessed 26 June 2021
[24] Koulouzis, S., et al.: Time-critical data management in clouds: Challenges and a dynamic real-time infrastructure planner (DRIP) solution. Concurrency and Computatation Practice and Experience 32(16), e5269 (2019)
[25] The default workflow of Laserchicken. Available at: https://laserchicken.readthedocs.io/en/latest/. Accessed 27 June 2021

AUTHOR BIOGRAPHY

Yuandou Wang is currently a Ph.D. student at the University of Amsterdam, the Netherlands. Her research interests include cloud computing, workflow scheduling, and decentralized workflow management.
ORCID: 0000-0003-4694-9572

Spiros Koulouzis obtained his Ph.D. from the University of Amsterdam, the Netherlands. Currently, he is a research associate at the LifeWatch VLIC. His research interests include distributed and parallel systems, workflow systems as well as scientific data management.
ORCID: 0000-0001-8652-315X

Riccardo Bianchi obtained his joint MS.c. in Big Data Engineering from the University of Amsterdam (UvA) in 2020. Currently, he is a research associate at the LifeWatch VLIC. His research interests include distributed and parallel systems, workflow systems as well as scientific data management.
ORCID: 0000-0002-8550-3824

Na Li is currently a Ph.D. student in the Informatics Institute at the University of Amsterdam, the Netherlands. Her research interests include research asset discovery, dataset search and data quality assessment.
ORCID: 0000-0001-7799-876X

Yifang Shi is a scientific developer for ecological applications of LiDAR Remote Sensing at LifeWatch-ERIC and the University of Amsterdam. Her research interest is in developing scientific workflows for ecological applications in biodiversity and ecosystem science using airborne and spaceborne LiDAR.
ORCID: 0000-0001-8770-8906

Joris Timmermans is Scientific Developer at the Lifewatch European infrastructure (ERIC), Virtual Laboratory Innovation Center (VLIC), as well as a guest researcher at the University of Amsterdam (UvA), the Netherlands, and at Leiden University (LU), the Netherlands. He presently works on developing essential biodiversity variables from remote sensing combining multi-sensor monitoring, data-intensive fusion techniques and ecology to track policy targets set for 2030.
ORCID: 0000-0002-6359-8092

W. Daniel Kissling is Associate Professor at the University of Amsterdam (UvA), the Netherlands. His research interest is in data-intensive biodiversity science, ecology and global biodiversity change, using large datasets, ecoinformatic tools, digital sensors and remote sensing.
ORCID: 0000-0002-7274-6755

Zhiming Zhao received his Ph.D. degree in Computer Science in 2004 from the University of Amsterdam (UvA). He is an assistant professor in Multiscale Network Systems (MNS) at UvA, and the technical manager in the Virtual Lab and Innovation Center (VLIC) of LifeWatch ERIC, a European research infrastructure in ecology and biodiversity science. His research focuses on data management, cloud computing, trustworthy service level agreement, and blockchain. He leads the UvA effort in several EU projects, including ARTICONF, CLARIFY, ENVRI-FAIR, and SWITCH projects. (https://staff.fnwi.uva.nl/z.zhao/)
ORCID: 0000-0002-6717-9418